INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Consensus Based on Strong Failure Detectors: Time and Message-Efficient Protocols

F. Greve, M. Hurfin, R. Macêdo et M. Raynal

N˚3855

Janvier 2000

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--3855--FR

RENNES

Consensus Based on Strong Failure Detectors: Time and Message-Efficient Protocols

F. Greve*, M. Hurfin*, R. Macêdo† et M. Raynal*

Thème 1 — Réseaux et systèmes
Projet ADP

Rapport de recherche n ° 3855 — Janvier 2000 — 12 pages

Abstract: The class of *strong* failure detectors (denoted $\mathcal{S}$) includes all failure detectors that suspect all crashed processes and that do not suspect some (a priori unknown) process that never crashes. So, a failure detector that belongs to $\mathcal{S}$ is intrinsically unreliable as it can arbitrarily suspect correct processes. Several $\mathcal{S}$-based consensus protocols have been designed. Some of them systematically require n computation rounds (n being the number of processes), each round involving n^2 or n messages. Others allow early decision (i.e., the number of rounds depends on the maximal number of crashes when there are no erroneous suspicions) but require each round to involve n^2 messages.

This paper presents an early deciding $\mathcal{S}$-based consensus protocol each round of which involves $3(n-1)$ messages. So, the proposed protocol is particularly time and message-efficient. Moreover, it can easily be generalized to reduce the number of rounds at the price of an increase in the number of messages per round.

Key-words: Asynchronous Distributed System, Consensus, Crash Failure, Perpetual Accuracy Property, Unreliable Failure Detector.

(Résumé : tsvp)

This work has been partially supported by a CNET-FRANCE TELECOM grant 98 1B 123 and by CNPq/Brazil grants.

* {Fabiola.Greve | Michel.Hurfin | Michel.Raynal}@irisa.fr
† LaSiD-CPD-UFBA, Campus de Ondina, CEP 40170-110 Bahia, Brazil. macedo@ufba.br

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Résoudre le consensus à l'aide de détecteurs de défaillances forts: Des protocoles efficaces en temps et en nombre de messages

Résumé : La classe des détecteurs de défaillances *forts* (dénotée $\mathcal{S}$) comprend tous les détecteurs de défaillances qui ont pour propriétés de suspecter les processus défaillants mais de ne pas suspecter au moins un des processus non défaillants (sans que l'identité de celui-ci soit connu a priori). De fait, un détecteur de défaillances appartenant à la classe $\mathcal{S}$ est intrinsèquement non fiable puisqu'il peut suspecter arbitrairement des processus corrects. Plusieurs protocoles de consensus fondés sur la classe $\mathcal{S}$ ont été conçus. Certains d'entre eux requièrent systématiquement n étapes de calcul (n étant le nombre de processus), chaque étape nécessitant l'émission de n ou n^2 messages. D'autres protocoles permettent une prise de décision au plus tôt (dans ce cas, le nombre d'étapes de calcul dépend du nombre maximal de défaillances lorsqu'il n'y a pas de suspicion erronée) mais requièrent n^2 messages à chaque étape de calcul.

Cet article présente un protocole de consensus fondé sur la classe $\mathcal{S}$ permettant des décisions au plus tôt. Chaque étape de calcul nécessite $3(n-1)$ messages. De fait, le protocole proposé est particulièrement efficace en temps et en nombre de messages. De plus, il peut aisément être généralisé afin de réduire le nombre d'étapes de calcul au prix d'une augmentation du nombre de messages échangés par étape.

Mots-clé : systèmes répartis asynchrones, consensus, pannes franches, propriété de précision perpétuelle, détecteurs de défaillances non fiables.

1 Introduction

Several crucial practical problems (such as *atomic broadcast* and *atomic commit*) encountered in the design of reliable applications built on top of unreliable asynchronous distributed systems, actually belong to a same family: the family of *agreement problems*. This family can be characterized by a single problem, namely the *Consensus* problem, that is their "*greatest common subproblem*". That is why the consensus problem is considered as a fundamental problem. This is practically and theoretically very important. From a practical point of view, this means that any solution to consensus can be used as a building block on top of which solutions to particular agreement problems can be designed. From a theoretical point of view, this means that an agreement problem cannot be solved in systems where consensus cannot be solved.

Informally, the consensus problem can be defined in the following way. Each process proposes a value and all correct processes have to decide the same value, which has to be one of the proposed values. Solving the consensus problem in asynchronous distributed systems where processes may crash is far from being a trivial task. It has been shown by Fischer, Lynch and Paterson [3] that there is no deterministic solution to the consensus problem in those systems as soon as processes (even only one) may crash. This impossibility result comes from the fact that, due to the uncertainty created by asynchrony and failures, it is impossible to precisely know the system state. So, to be able to solve agreement problems in asynchronous distributed systems, those systems have to be "augmented" with additional assumptions that make consensus solvable in such improved systems. A major and determining advance in this direction has been done by Chandra and Toueg who have proposed [1] (and investigated with Hadzilacos [2]) the *Unreliable Failure Detector* concept.

A failure detector can informally be seen as a set of *oracles*, one per process. The failure detector module (oracle) associated with a process provides it with a list of processes it guesses to have crashed. A failure detector can make mistakes by not suspecting a crashed process, or by erroneously suspecting a correct process. In their seminal paper [1], Chandra and Toueg have defined two types of property to characterize classes of failure detectors. A class is defined by a *Completeness* property and an *Accuracy* property. A completeness property is on the actual detection of crashes. The completeness property we are interested in basically states that "every crashed process is eventually suspected by every correct process". An accuracy property limits the mistakes a failure detector can make.

In this paper, we are interested in solving the consensus problem in asynchronous distributed systems equipped with a failure detector of the class $\mathcal{S}$. A failure detector of this class suspects all crashed processes (completeness) and guarantees that there is a correct process that is never suspected, but this process is not a priori known (perpetual weak accuracy). Several $\mathcal{S}$-based consensus protocols have been proposed. They all assume $f \leq n-1$ (where f is the maximal number of processes that may crash), and consequently are optimal with respect to the number of crash failures they can tolerate. They all proceed in asynchronous "rounds".

The $\mathcal{S}$-based consensus protocol proposed in [1] requires exactly n rounds, each round involving n^2 messages (each message being composed of n values). The $\mathcal{S}$-based protocols presented in [6, 7] also require n rounds, but each round involves only n messages carrying a single value. It is important to emphasize that these three protocols require n rounds whatever the value of f, the number of actual crashes and the occurrences of erroneous suspicions are.

To our knowledge, very few *early deciding* $\mathcal{S}$-based consensus protocols have been proposed, more precisely, we are only aware of the generic protocol presented in [5][1]. When instantiated with a failure detector $\in \mathcal{S}$, this generic protocol provides a $\mathcal{S}$-based consensus protocol that terminates in at most $(f+1)$ rounds when there are no erroneous suspicions. So, when the failure detector is tuned to very seldom make mistakes, this protocol provides early decision. Each round of this protocol involves n^2 messages (each message being made of a proposed value plus a round number) and one or two communication steps.

This paper presents an *early deciding* $\mathcal{S}$-based consensus protocol. When there are no erroneous suspicions, the proposed protocol requires $(f+1)$ rounds, in the worst case. When there are neither crashes nor erroneous suspicions, it requires a single round, a round being made up of two communication steps. Each round involves $3(n-1)$ messages, and each message carries at most three values: a round number, a proposed value and a timestamp (i.e., another round number). So, the protocol is both time and message-efficient. Moreover, a generalization of the protocol exhibits an interesting tradeoff between the number of rounds and the number of messages per round.

The paper is made up of five sections. Section 2 introduces the asynchronous system model, the class $\mathcal{S}$ of failure detectors, and the consensus problem. Then, Section 3 presents the $\mathcal{S}$-based consensus protocol and proves it is correct. Section 4 discusses its cost and presents the generalization that can reduce the number of rounds at the price of an increase in the number of messages per round. Finally, Section 5 concludes the paper.

2 Asynchronous Distributed Systems, Failure Detectors and the Consensus Problem

The system model is patterned after the one described in [1, 3]. A formal introduction to failure detectors is provided in [1].

2.1 Asynchronous Distributed System with Process Crash Failures

We consider a system consisting of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. A process can fail by *crashing*, (i.e., by prematurely halting). It behaves correctly (i.e., according to its specification) until it (possibly) crashes. By definition, a *correct* process is a process that does not crash. Let f denote the maximum number of processes that can crash ($f \leq n-1$).

Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are not required to be FIFO, they may also duplicate messages. They are only assumed to be reliable in the following sense: they do not create, alter or lose messages. This means that a message sent by a process p_i to a process p_j is assumed to be eventually received by p_j, if p_j is correct[2].

The multiplicity of processes and the message-passing communication make the system *distributed*. There is no assumption about the relative speed of processes or the message transfer delays. This absence of timing assumptions makes the distributed system *asynchronous*.

[1] The generic dimension of the protocol introduced in [5] lies in the class of the failure detector it relies on. This generic protocol can be instantiated with any failure detector of $\mathcal{S}$ (provided $f \leq n-1$) or $\diamond\mathcal{S}$ (provided $f < n/2$). A failure detector that belongs to $\diamond\mathcal{S}$: (1) eventually suspects permanently all crashes processes, and (2) guarantees that there is a time after which there is a correct process that is never suspected.

[2] The "no message loss" assumption is required to ensure the Termination property of the protocol. The "no creation and no alteration" assumptions are required to ensure its Validity and Agreement properties.

2.2 The Class $\mathcal{S}$ of Unreliable Failure Detectors

Informally, a failure detector consists of a set of modules, each attached to a process: the module attached to p_i maintains a set (named $suspected_i$) of processes it currently suspects to have crashed. Any failure detector module is inherently unreliable: it can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct one. Moreover, suspicions are not necessarily stable: a process p_j can be added to and removed from a set $suspected_i$ according to whether p_i's failure detector module currently suspects p_j or not. As in [1], we say "process p_i suspects process p_j" at some time t, if at time t we have $j \in suspected_i$.

As indicated in the introduction, a failure detector class is defined by two abstract properties, namely a *Completeness* property and an *Accuracy* property. In this paper we are interested in the following properties [1]:

- Strong Completeness: Eventually, every crashed process is permanently suspected by every correct process.
- Perpetual Weak Accuracy: Some correct process is never suspected.

The failure detectors that satisfy these properties define the class $\mathcal{S}$ (*Strong* failure detectors). It is important to note that a failure detector $\in \mathcal{S}$ can make an arbitrary number of mistakes: at any time all (but one) correct processes can be erroneously suspected. Moreover, a process can alternatively suspect and not suspect some correct processes.

2.3 The Consensus Problem

In the consensus problem, every correct process p_i *proposes* a value v_i and all correct processes have to *decide* on some value v, in relation with the set of proposed values. More precisely, the *Consensus* problem is defined by the three following properties [1, 3]:

- Termination: Every correct process eventually decides on some value.
- Validity: If a process decides v, then v was proposed by some process.
- Agreement: No two correct processes decide differently.

The agreement property applies only to correct processes. So, it is possible that a process decides on a distinct value just before crashing. *Uniform Consensus* prevents such a possibility. It has the same Termination and Validity properties plus the following agreement property:

- Uniform Agreement: No two processes (correct or not) decide differently.

In the following we are interested in the *Uniform Consensus* problem.

3 The $\mathcal{S}$-Based Consensus Protocol

This section first presents a consensus protocol for an asynchronous distributed system augmented with a failure detector of the class $\mathcal{S}$ (Section 3.1). Then, it presents its correctness proof (Section 3.2).

3.1 The Protocol

Underlying Principles As other failure detector-based consensus protocols, the proposed protocol uses the *rotating coordinator* paradigm and processes proceed in asynchronous rounds [1]. There are at most n rounds. Each round r ($1 \leq r \leq n$) is managed by a predetermined coordinator, namely, p_r. Moreover, during r, the coordinator of the next round (namely, p_{r+1}) acts also a particular role. Each process p_i manages three local variables: the current round number (r_i), its current estimate of the decision value (est_i), and a timestamp (ts_i) that indicates the round number during which it adopted its current estimate est_i.

As in [5, 6, 7], during a round, the current coordinator tries to impose its current estimate as the decision value. To attain this goal, each round r is made of two steps (see Figure 1).

- During the first step (lines 4-6) the current coordinator p_r broadcasts a message carrying its current estimate, namely, the message PHASE1(r, est_r).

  When a process p_i receives such a PHASE1(r, v) message, it adopts v as its new estimate and consequently updates ts_i to the current round number (line 6). If p_i suspects p_r, its "state variables" est_i and ts_i keep their previous values.

- During the second phase (lines 7-13), each process sends its "current state" to the current round coordinator (p_r) and the next round coordinator (p_{r+1}). This "state" is carried by a PHASE2 message (line 8). The triple (r, est_i, ts_i) indicates that during r, (1) the estimate of the decision value considered by p_i is est_i and (2) this value has been adopted during the round ts_i.

  Then, p_r and p_{r+1} follow the same behavior: each waits until it has received PHASE2 messages from all the processes it does not suspect (let us note that due to the completeness property of the underlying failure detector, all crashed processes are eventually suspected).

  If all the PHASE2 messages the process p_r (resp. p_{r+1}) has received have a timestamp equal to the current round number (r), p_r (resp. p_{r+1}) decides on its current estimate (lines 11-12). This means that the current estimate of p_r has been imposed as decision value.

  Whether the process p_{r+1} decides during r or proceeds to $r+1$, it must have a correct estimate (in order not to violate the consensus agreement property). This is ensured by requiring it to update its local estimate (est_{r+1}) to the estimate it has received with the highest timestamp in a PHASE2(r, est, ts) message (line 10).

Structure The protocol is fully described in Figure 1. A process p_i starts a consensus execution by invoking Consensus(v_i), where v_i is the value it proposes. The protocol terminates for p_i when it executes the statement *return* which provides it with the decided value (at line 12 or 15).

It is possible that distinct processes do not decide during the same round. To prevent a process from blocking forever (i.e., waiting for a value from a process that has already decided), a process that decides, uses a reliable broadcast [4] to disseminate its decision value. To this end, the Consensus function is made of two tasks, namely, $T1$ and $T2$. $T1$ implements the previous discussion. Line 12 and $T2$ implement the reliable broadcast.

3.2 Proof

3.2.1 Validity

Theorem 1 *If a process p_i decides v, then v was proposed by some process.*

```
Function Consensus(v_i)

Task T1:
(1)  r_i ← 0; est_i ← v_i; ts_i ← 0;
(2)  while r_i < n do
(3)      r_i ← r_i + 1;
         % p_{r_i} is the coordinator of the current round; p_{r_i+1} is the coordinator of the next round %

         ———————— Phase 1 of round r: p_{r_i} proposes to all ————————————————————-
(4)      if (i = r_i) then ∀j: send PHASE1(r_i, est_i) to p_j endif;
(5)      wait until (PHASE1(r_i, v) has been received from p_{r_i} ∨ r_i ∈ suspected_i);
(6)      if (PHASE1(r_i, v) received from p_{r_i}) then est_i ← v; ts_i ← r_i endif;

         ———————— Phase 2 of round r: each process replies to p_{r_i} and p_{r_i+1} ——————-
(7)      let X = {r_i, r_i + 1} if (r_i < n), X = {r_i} otherwise;
(8)      ∀j ∈ X: send PHASE2(r_i, est_i, ts_i) to p_j;
(9)      if (i ∈ X) then wait until (PHASE2(r_i, est, ts) received from all non suspected processes);
(10)                    if (i = r_i + 1) then est_i ← est received with the highest ts endif;
(11)                    if (all PHASE2 messages are such that ts = r_i)
(12)                                then ∀j ≠ i : send DECISION(est_i) to p_j; return(est_i) endif
(13)     endif
(14) endwhile

Task T2:
(15) upon the reception of DECISION(est) from p_k: ∀j ≠ i, k : send DECISION(est)to p_j; return(est)
```

Figure 1: The $\mathcal{S}$-Based Consensus Protocol

Proof The proof is by induction on the round number. Initially (round $r = 0$), all est_i local variables contain proposed values (line 1). Let us assume that all estimates contain proposed values at the end of $r - 1$. We show they contain proposed values at the end of r.

An est_i local variable is updated at line 6 or at line 10. At line 6, est_i is updated to the estimate of the round r coordinator, which is the estimate value this process had at the end of $r - 1$. By the induction assumption, this is a proposed value. So, at the end of the first phase of r, all estimate values contain proposed values.

During the second phase of r, if est_i is updated at line 10, it takes the value of an estimate that has been updated during some round ts. If $ts < r$, this update occurred during the first or the second phase of ts. If $ts = r$, this update occurred during the first phase of r. Hence, at the end of r, each est_i variable contains a value initially proposed by a process.

As a decided value is the value of an est_i variable (lines 12 and 15), the lemma follows. $\square_{Theorem\ 1}$

3.2.2 Termination

Lemma 1 *If no process decides during $r' \leq r$, then all correct processes start $r + 1$.*

Proof The proof is by contradiction. Let us assume that no process decided during a round $r' < r$, where r is the smallest round number during which a correct process p_i blocks forever. So, p_i is blocked at line 5 or 9 (in a **wait** statement).

Let us first examine the case where p_i blocks at line 5. If $i = r$, it cannot block at line 5: as it broadcast a message at line 4, it receives it. If $i \neq r$, then (1) either p_i suspects p_r, (2) or p_i never

suspects p_r. In case (1), p_i cannot block. In case (2), due to the strong completeness property, this means that p_r is correct. From the previous observation, p_r has broadcast its current estimate at line 4, and p_i eventually receives it. It follows that during round r no correct process p_i remains blocked forever at line 5. Consequently, (A) all correct processes send a PHASE2 message at line 8.

Let us now consider the case where p_i blocks forever at line 9; so p_i is p_r or p_{r+1}. Due to (A) and the strong completeness property of the failure detector (namely, all crashed processes are eventually suspected), it follows that if p_r (resp. p_{r+1}) is correct, it receives PHASE2 messages from all the processes it does not suspect. Hence, no correct process can block forever at line 9 during r.

As no correct process blocks forever during round r, they all proceed to the round $r+1$. A contradiction. $\square_{Lemma\ 1}$

Theorem 2 *If a process p_i is correct, then it decides.*

Proof If a (correct or not) process decides, then, due to the sending of DECISION messages at line 12 or at line 15, and the no message loss property, any correct process will receive such a message and decide accordingly (line 15).

So, suppose that no process decides. The proof is by contradiction. Due to the accuracy property of the underlying failure detector, there is a correct process (say, p_x) that is never suspected. Moreover, let r be the round that is coordinated by p_x. As by assumption no process decides, due to Lemma 1, all the correct processes eventually start round r.

The process p_x starts round r by broadcasting its current estimate value est_x. As p_x is not suspected, any process p_i that participates in round r (this set includes all the correct processes) receives PHASE1(r, est_x) at line 5, and executes the updates $est_i \leftarrow est_x$ and $ts_i \leftarrow r$. If follows that all messages sent at line 8 are PHASE2(r, est_x, r). Hence, due to the no message loss property, p_x receives only PHASE2(r, est_x, r) messages at line 9, and consequently decides at lines 11-12. A contradiction. $\square_{Theorem\ 2}$

3.2.3 Uniform Agreement

Lemma 2 *Let $r \geq 1$ be the first round during which a process p_x that is never suspected sends* PHASE2(r, v, r)[3]. *We have:*
i). No process has decided before r.
ii). Let r' be any round $\geq r$. If $p_{r'}$ sends a PHASE1 *message, then this message carries v (i.e., it is* PHASE1(r', v)*).*

Proof The proof of (i) follows from the definition of r ("first" round during which p_x ...) and the fact that, to decide during the round r, a process has to receive PHASE2 messages carrying a timestamp equal to r from all the processes it does not suspect, hence from p_x.

The proof of (ii) is by induction on the round number.

- Base case: $r' = r$. By assumption r is the first round during which a process p_x that is never suspected "echoes" at line 8 (by sending PHASE2(r, v, r)) the PHASE1(r, v) message it has received. Hence, the lemma trivially holds.

[3]This means r is the first round during which p_x receives a value v carried by a PHASE1 message and consequently echoes it by sending PHASE2 messages. Using the traditional terminology, from that round v is "locked" and consequently will be the decided value.

- Induction case: $r' > r$. Let us assume the lemma holds for any round k such that $r \leq k \leq r'$. We show it is true for $r'+1$.

  Due to the induction assumption, all PHASE1(k, w) messages with $r \leq k \leq r'$ are such that $w = v$. Let us consider two sets of processes:

  - The set R. This set includes all processes p_j that have received a PHASE1(k, w) message with $r \leq k \leq r'$. $\forall p_j \in R$: p_j updated est_j to v and ts_j to a round number k' such that $r \leq k' \leq r'$.
  - The set Q. This set includes all processes p_j that have not received a PHASE1(k, w) message with $r \leq k \leq r'$. Consequently, $\forall p_j \in Q$: ts_j has a value $< r$.

  Let us now consider the behavior of $p_{r'+1}$ during the second phase of round r'. It first receives PHASE2 messages from all the processes it does not suspect (line 9). Then (line 10), $p_{r'+1}$ updates $est_{r'+1}$ to the est received with the highest timestamp. As the set of processes from which $p_{r'+1}$ received PHASE2 messages includes at least one process of R (namely, p_x), due to the previous observation, the highest timestamp contained in those PHASE2 messages cannot come from a process $\in Q$: it necessarily comes from a process of R. Hence, $est_{r'+1}$ is set to v. It follows that if, during the round $r'+1$, $p_{r'+1}$ broadcasts a PHASE1 message, this message carries the pair $(r'+1, v)$.

$\square_{Lemma\ 2}$

Lemma 3 *If during a round r a process p_i receives* PHASE2(r, v, r)*, then the coordinator of r has sent* PHASE1(r, v).

Proof The proof of this lemma directly follows from the first phase of the protocol. $\square_{Lemma\ 3}$

Theorem 3 *No two processes decide differently.*

Proof If a process decides a value at line 15, then this value has been decided by another process at line 12. So, we only consider values decided at line 12.

Let p_i and p_j be two processes that decide $v1$ and $v2$ during the rounds $r1$ and $r2$, respectively. As p_i (resp. p_j) decides during the round $r1$ (resp. $r2$), it has received PHASE2$(r1, v1, r1)$ (resp. PHASE2$(r2, v2, r2)$) during $r1$ (resp. $r2$). Due to Lemma 3, the coordinator of $r1$ (resp. $r2$) has sent PHASE1$(r1, v1)$ (resp. PHASE1$(r2, v2)$). Without loss of generality let us assume $r1 \leq r2$. Moreover, let r be the round characterized in Lemma 2. We conclude from Lemma 2 that the messages PHASE1(r, v), PHASE1$(r1, v1)$ and PHASE1$(r2, v2)$ are such that $r \leq r1 \leq r2$ and $v = v1 = v2$.

$\square_{Theorem\ 3}$

4 Discussion

4.1 Cost of the Protocol

Time complexity The number of rounds of the protocol is $\leq n$. Differently from the $\mathcal{S}$-based consensus protocols described in [1, 6, 7] that always require n rounds, the actual number of rounds of the proposed protocol depends on failures occurrences and erroneous suspicions occurrences. So, to analyze the time complexity of the protocol, we consider the length of the sequence of messages

(number of communication steps) exchanged during a round. Moreover, as we do not master the quality of service offered by the underlying failure detector, but as in practice failure detectors can be tuned to very seldom make mistakes, we do this analysis considering the underlying failure detector behaves reliably. In such a context, the time complexity of a round is characterized by a pair of integers [5]. Considering the most favorable scenario that allows to decide during the current round, the first integer measures its number of communication steps (without counting the cost of the reliable broadcast implemented by the task $T2$). The second integer considers the case where a decision cannot be obtained during the current round and measures the minimal number of communication steps required to progress to the next round. Let us consider these scenarios.

- The first scenario is when the current round coordinator is correct and is not suspected. In that case, 2 communication steps are required to decide. During the first step, the current coordinator broadcasts a PHASE1 message (line 4). During the second step, each process sends a PHASE2 message (line 8). So, in the most favorable scenario that allows to decide during the current round, the round is made up of two communication steps.

- The second scenario is when the current round coordinator has crashed and is suspected by all processes. In that case, as processes correctly suspect the coordinator (line 5), the first communication step is actually skipped. Processes only send PHASE2 message (line 8) and proceed to the next round. So, in the most favorable scenario to proceed to the next round, the round is made up of a single communication step.

So, when the underlying failure detector behaves reliably, according to the previous discussion, the time complexity of a round is characterized by the pair $(2, 1)$ of communication steps.

Message complexity of a round During each round, the round coordinator broadcasts a PHASE1 message and each process sends two PHASE2 messages. Hence, the message complexity of a round is bounded by $3(n-1)$.

Message type and size There are three types of message: PHASE1, PHASE2 and DECISION. A DECISION message carries only a proposed value. A PHASE1 message carries a proposed value plus a round number. A PHASE2 message carries a proposed value plus two round numbers. As the number of rounds is bounded by n, the size of the round number is bounded by $log_2(n)$.

Let $|v|$ be the bit size of a proposed value. According to the previous discussion, $3n(n-1)(|v| + log_2 n)$ is an upper bound of the bit complexity of the protocol.

Comparison with related work Table 1 compares the $\mathcal{S}$-based consensus protocols that we know ("nb", "msg" and "c_s" are shortcuts for "number", "messages" and "communication steps", respectively). The protocols of the first two lines systematically require n rounds. [6] and [7] each proposes a particular generalization of $\mathcal{S}$; we consider here their protocols used with $\mathcal{S}$. As previously noticed, the behavior of the protocols of the last two lines depends on failure occurrences and false failure suspicions (that is why they allow early decision in "good" scenarios). For those lines, the table considers that the failure detector does not make mistakes, there are f failures, and there is one failure per round. For the generic protocol described in [5], we consider its instantiation with $\mathcal{S}$.

Protocol	early deciding	total nb of rounds	max nb of c_s/round	max nb of msg/round	size of the biggest msg
[1]	no	n	1	$n(n-1)$	$n\|v\| + log_2 n$
[6], [7]	no	n	1	$(n-1)$	$\|v\|$
[5]	yes	$(f+1)$	2	$n(n-1)$	$\|v\| + log_2 n$
Proposed prot.	yes	$(f+1)$	2	$3(n-1)$	$\|v\| + 2log_2 n$

Table 1: Comparison of $\mathcal{S}$-Based Consensus Protocols

4.2 Revisiting the Second Phase

The section proposes a modification that can reduce the number of rounds at the price of an increase in the number of messages per round. This modification simply consists in allowing each round to define the set of processes that, during the second phase, receive and process PHASE2 messages. The modified protocol is obtained by replacing the line 7 of Figure 1 by **let** X =check_pred(r_i). The second phase resulting from this modification is described in Figure 2.

described in Figure 2.

```
(7)  let X =check_pred(r_i); % X: set of process identities including r_i and r_i + 1 (if ≤ n) %
(8)  ∀j ∈ X: send PHASE2(r_i, est_i, ts_i) to p_j;
(9)  if (i ∈ X) then wait until (PHASE2(r_i, est, ts) received from all non suspected processes);
(10)                 if (i ≠ r_i) then est_i ← est received with the highest ts endif;
(11)                 if (all PHASE2 messages are such that ts = r_i)
(12)                                 then ∀j ≠ i : send DECISION(est_i) to p_j; return(est_i) endif
(13) endif
```

Figure 2: Phase 2: Revisited

More precisely, let X_r be the set of processes that, during the round r, test the decision predicate (line 11). In the original protocol, X_r always includes exactly two processes, namely, $X_r = \{r, r+1\}$ (except X_n which includes only n). In the modified protocol, this set is defined by a deterministic[4] function X_r =check_pred(r), that always includes (but is not limited to) $\{r, r+1\}$; moreover, the cardinality of X_r can vary according to r. The reader can check this modification leaves the protocol correct.

This modification can allow processes to decide "directly" (i.e., before receiving a DECISION message). For a round r, the number of PHASE2 messages now depends on the cardinality of X_r. During r, the number of PHASE2 messages is $(n-1)(|X_r|+1)$.

When, $\forall r$, check_pred$(r) = \{1, \cdots, n\}$, we get a $\mathcal{S}$-based consensus protocol that requires $O(n^2)$ messages per round, and is very close to the protocol obtained by instantiating with $\mathcal{S}$ the generic protocol described in [5].

5 Conclusion

The paper has studied the consensus problem in the setting of asynchronous distributed system equipped with a failure detector of the class $\mathcal{S}$. This class includes all the failure detectors that

[4]This function has to be deterministic in order the same output be produced when it is invoked by different processes during the same round.

suspect all crashed processes and that do not suspect some (a priori unknown) correct process. The proposed protocol proceeds in asynchronous rounds and allows early decision. If there are neither failures nor false suspicions the decision is obtained in a single round. A round is made up of two communication steps and involves $3(n-1)$ messages.

The proposed protocol compares very favorably to the previous $\mathcal{S}$-based consensus protocols, as those require n rounds or n^2 messages per round.

References

[1] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, March 1996.

[2] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[3] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, April 1985.

[4] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press (S. Mullender Ed.), New-York, pp. 97-145, 1993.

[5] Mostéfaoui A. and Raynal M., Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a General Quorum-Based Approach. *Proc. 13th Int. Symposium on Distributed Computing (DISC'99) (formerly, WDAG)*, Springer-Verlag LNCS 1693, pp. 49-63, (P. Jayanti Ed.), Bratislava (Slovaquia), September 1999.

[6] Mostéfaoui A. and Raynal M., Consensus Based on Failure Detectors with a Perpetual Weak Accuracy Property. *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS'2k), (14th IPPS/11th SPDP)*, Cancun (Mexico), May 2000.

[7] Yang J., Neiger G. and Gafni E., Structured Derivations of Consensus Algorithms for Failure Detectors. *Proc. 17th ACM Symposium on Principles of Distributed Computing*, Puerto Vallarta (Mexico), pp.297-308, 1998.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399